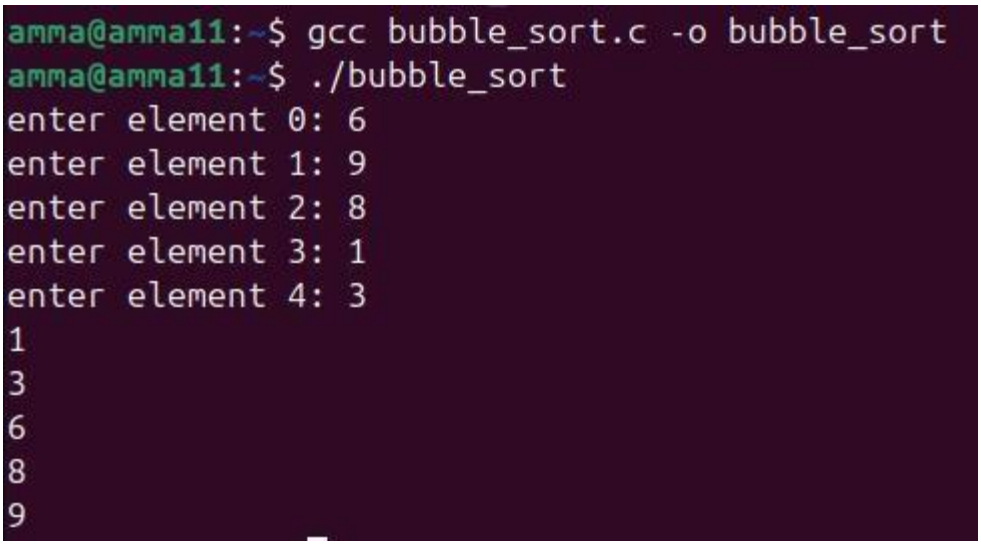
Design and Analysis of Algorithms

Week-2 Assignment

1. Bubble Sort

PROGRAM:

```
#include <stdio.h>
int main() {
int n;
printf("ENTER NO.OF.ELEMENTS:");
scanf("%d",&n);
int arr[n];
printf("ENTER THE VALUES:");
for(int i=0;i < n;i++){
scanf("%d",&arr[i]);
}
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("\t%d\t,",arr[i]);
    }
    return 0;
}
```



TIME COMPLEXITY:- $O(n^2)$
JUSTIFICATION:-
Outer loop runs n times.
Inner loop runs up to n times.
Total operations ≈ n × n.
SPACE COMPLEXITY:- O(1)
JUSTIFICATION:-
int n → 4 bytes
int i → 4 bytes
int j → 4 bytes
int temp → 4 bytes
Only constant extra space is used.

2. Insertion Sort

PROGRAM:

```
#include <stdio.h>
int main() {
    int n;
    printf("ENTER NO.OF ELEMENTS");
    scanf("%d",&n);
    int arr[n];
    printf("ENTER THE ELEMENTS:");
    for(int i=0; i <n;i++){
    scanf("%d",&arr[i]);
    }
    for (int i = 1; i < n; i++) {
```

```
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j] > key) {
    arr[j + 1] = arr[j];
```

```
            j--;
        }
        arr[j + 1] = key;
    }
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

```
amma@amma11:~$ gcc insertion.c -o insertion
amma@amma11:~$ ./insertion
enter element 0: 7
enter element 1: 9
enter element 2: 1
enter element 3: 0
enter element 4: 4
0
1
4
7
9
```

TIME COMPLEXITY:- $O(n^2)$

JUSTIFICATION:-

Outer loop runs n times.

Inner while loop runs up to n times.

Total operations ≈ n × n.

SPACE COMPLEXITY:- O(1)

JUSTIFICATION:-

int n → 4 bytes
int i → 4 bytes
int j → 4 bytes
int key → 4 bytes
Only constant extra space is used.

3. Selection Sort

PROGRAM:

```
#include <stdio.h>
int main() {
    int n;
    printf("ENTER NO. OF ELEMENTS: ");
    scanf("%d", &n);
    int arr[n];
    printf("ENTER THE VALUES: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
```

```
    }
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

```
amma@amma11:~$ gcc selection_sort.c -o selection_sort
amma@amma11:~$ ./selection_sort
enter element 0: 9
enter element 1: 1
enter element 2: 0
enter element 3: 3
enter element 4: 6
0
1
3
6
9
```

TIME COMPLEXITY:- $O(n^2)$

JUSTIFICATION:-

Outer loop runs n times.

Inner while loop runs up to n times.

Total operations ≈ n × n.

SPACE COMPLEXITY:- O(1)

JUSTIFICATION:-

int n → 4 bytes

int i → 4 bytes

int j → 4 bytes

int key → 4 bytes
Only constant extra space is used.

4. Bucket Sort

PROGRAM:

```
#include <stdio.h>
```

```
int main() {
    int n;
    printf("ENTER NO. OF ELEMENTS: ");
    scanf("%d", &n);
    int arr[n];
    printf("ENTER THE VALUES (0 to 100): ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int bucket[101] = {0};
    for (int i = 0; i < n; i++) {
        bucket[arr[i]]++;
    }
    printf("Sorted array: ");
    for (int i = 0; i <= 100; i++) {
        while (bucket[i] > 0) {
            printf("%d ", i);
            bucket[i]--;
        }
    }
    return 0;
}
```

```
amma@amma11:~$ gcc bucket_sort.c -o bucket_sort
amma@amma11:~$ ./bucket_sort
enter element 0: 8
enter element 1: 2
enter element 2: 0
enter element 3: 1
enter element 4: 7
0
1
2
7
8
```

TIME COMPLEXITY:- O(n + k)

JUSTIFICATION:-

Elements are distributed into k buckets.

Each bucket is sorted individually.

Average case operations ≈ n + k.

SPACE COMPLEXITY:- O(n + k)

JUSTIFICATION:-

Buckets array → extra space

int n → 4 bytes

int i → 4 bytes

Additional space for buckets is required.

5. Heap Sort

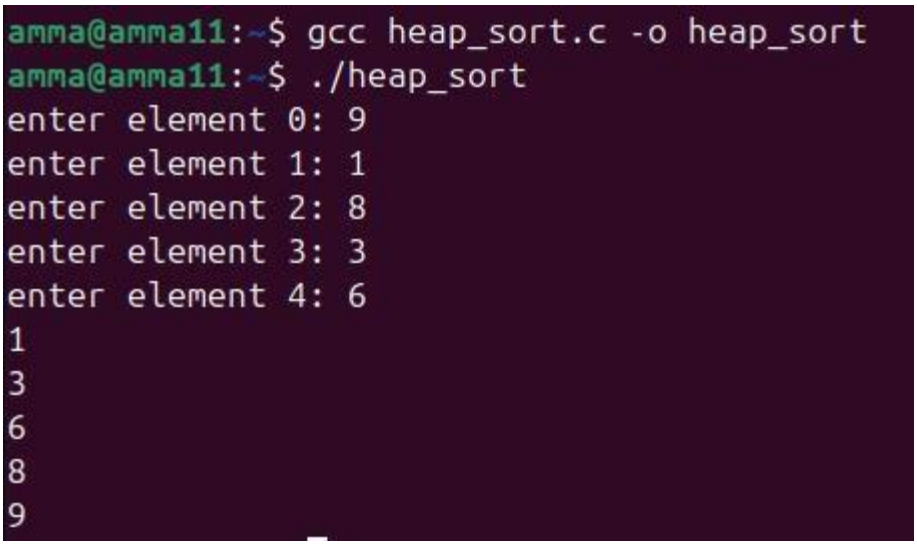
PROGRAM:

```
#include<stdio.h>

int main() {

    int n;

    printf("ENTER NO. OF ELEMENTS: ");
```

```
scanf("%d", &n);
int arr[n];
printf("ENTER THE VALUES: ");
for (int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}
for (int i = 1; i < n; i++) {
    int child = i;
    while (child > 0) {
        int parent = (child - 1) / 2;
```

```
        if (arr[parent] < arr[child]) {
            int temp = arr[parent];
            arr[parent] = arr[child];
            arr[child] = temp;
            child = parent;
        } else {
            break;
        }
    }
}
for (int i = n - 1; i > 0; i--) {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    int parent = 0;
    while (1) {
        int left = 2 * parent + 1;
        int right = 2 * parent + 2;
        int largest = parent;
        if (left < i && arr[left] > arr[largest])
            largest = left;
        if (right < i && arr[right] > arr[largest])
```

```
            largest = right;
        if (largest != parent) {
            int temp2 = arr[parent];
            arr[parent] = arr[largest];
            arr[largest] = temp2;
            parent = largest;
        } else {
            break;
        }
    }
  }
  printf("Sorted array: ");
  for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
  }
  return 0;
}
```



TIME COMPLEXITY:- O(n log n)

JUSTIFICATION:-

Building heap takes O(n).

Heapify operation runs log n for each element.

Total operations ≈ n log n.

SPACE COMPLEXITY:-O(1)

JUSTIFICATION:-

int n → 4 bytes

int i → 4 bytes

int temp → 4 bytes

Sorting is done in-place.

6.BFS

Code:

#include <stdio.h>

```
// CH.SC.U4CSE24133 - Min Heap

void minHeapify(int a[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    int temp;

    if (left < n && a[left] < a[smallest])
        smallest = left;

    if (right < n && a[right] < a[smallest])
        smallest = right;

    if (smallest != i) {
        temp = a[i];
        a[i] = a[smallest];
        a[smallest] = temp;

        minHeapify(a, n, smallest);
    }
}

void buildMinHeap(int a[], int n) {
    int i;
    for (i = n / 2 - 1; i >= 0; i--)
        minHeapify(a, n, i);
}

int main() {
```

```
    printf("CH.SC.U4CSE24133\n");

    int a[6] = {3, 9, 2, 1, 4, 5};
    int n = 6, i;

    buildMinHeap(a, n);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

```
BFS Traversal: 2 root@amma52:/home/amma/Documents# ./bfs
Enter number of nodes: 3
Enter adjacency matrix:
0 1 0 1 1 1 1 1 1
Enter starting node: 0

BFS Traversal: 0 1 2 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(V + E)

JUSTIFICATION:-

Each vertex is visited once → O(V)

Each edge is explored once → O(E)

Total operations ≈ V + E

SPACE COMPLEXITY:- O(V)

JUSTIFICATION:-

Queue can store up to V vertices

Visited array of size V

Only linear extra space is used

7.DFS
Code:

```
#include <stdio.h>

#define MAX 100

int visited[MAX] = {0};

void dfs(int graph[MAX][MAX], int n, int node) {
    printf("%d ", node);
    visited[node] = 1;

    for (int i = 0; i < n; i++) {
        if (graph[node][i] == 1 && !visited[i]) {
            dfs(graph, n, i);
        }
    }
}

int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    int graph[MAX][MAX];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
```

```
        }
    }

    int start;
    printf("Enter starting node: ");
    scanf("%d", &start);

    printf("\nDFS Traversal: ");
    dfs(graph, n, start);

    return 0;
}
```

Output:

```
BFS Traversal: 0 1 2 root@amma52:/home/amma/Documents# gcc -o dfs dfs.c
root@amma52:/home/amma/Documents# ./dfs
Enter number of nodes: 3
Enter adjacency matrix:
0 1 0 1 1 0 1 1 1
Enter starting node: 0

DFS Traversal: 0 1 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(V + E)

JUSTIFICATION:-

Each vertex visited once → V

Each edge visited once → E

SPACE COMPLEXITY:- O(V)

JUSTIFICATION:-

Recursion stack → V

Visited array → V

N.ASHWATH

CH.SC.U4CSE24133